Project: ZuluNation Motor

Introduction /Overview:

The HR department at Zulunation Motors wants to take some initiatives to improve employee satisfaction levels at the company. They collected data from employees, and They have the following question:

what's likely to make the employee leave the company?

Data Source:

The dataset is a fictitious example created for practice and knowledge.

Objective:

The goals in this project are to analyze the data collected by the HR department and to build a model/s that predicts whether or not an employee will leave the company. By successfully predicting which employees are likely to quit, it might be possible to identify factors that contribute to their decision to leave.

Setting up the environment, importing packages and load the dataset :

In [84]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data = pd.read_csv('C:/Users/engmo/OneDrive/Desktop/Projects to show of
```

EDA and Data cleaning

In [3]:
```
data.head()
```

Out[3]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_com
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

In [4]:

```
data.columns
```

Out[4]:

```
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_montly_hours', 'time_spend_company', 'Work_accident', 'lef
t',
       'promotion_last_5years', 'Department', 'salary'],
      dtype='object')
```

In [21]:

```
data.rename(columns={'average_montly_hours':'average_monthly_hours'},in
```

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64  
 3   average_montly_hours   14999 non-null  int64  
 4   time_spend_company     14999 non-null  int64  
 5   Work_accident          14999 non-null  int64  
 6   left                   14999 non-null  int64  
 7   promotion_last_5years  14999 non-null  int64  
 8   Department             14999 non-null  object 
 9   salary                 14999 non-null  object 
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

In [6]:

```
## lets standarize the column names
data.columns = [col.lower()for col in data.columns]
## lets rename some of the columns to improve simpilicity.
data.rename(columns={'time_spend_company':'tenure'},inplace=True)
## display the first 10 rows of the dataset
data.head(10)
```

Out[6]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	tenure	work_acc
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	
5	0.41	0.50	2	153	3	
6	0.10	0.77	6	247	4	
7	0.92	0.85	5	259	5	
8	0.89	1.00	5	224	5	
9	0.42	0.53	2	142	3	

In [7]:
```
# A descriptive analysis of the data.
data.describe()
```

Out[7]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	tenure
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000
mean	0.612834	0.716102	3.803054	201.050337	3.49823
std	0.248631	0.171169	1.232592	49.943099	1.46013
min	0.090000	0.360000	2.000000	96.000000	2.000000
25%	0.440000	0.560000	3.000000	156.000000	3.000000
50%	0.640000	0.720000	4.000000	200.000000	3.000000
75%	0.820000	0.870000	5.000000	245.000000	4.000000
max	1.000000	1.000000	7.000000	310.000000	10.000000

In [8]:
```
data['promotion_last_5years'].value_counts(normalize=True)
```

Out[8]:
```
0    0.978732
1    0.021268
Name: promotion_last_5years, dtype: float64
```

Check missing values:

In [9]:
```
data.isna().sum() #TO find the missing values in the dataset
```

Out[9]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
tenure                   0
work_accident            0
left                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

Check duplicates:

In [20]:
```
duplicates = data.duplicated() ## Inspect duplicates.
duplicates_rows = data[duplicates]  ## Creating a DataFrame For duplica
duplicates_rows.head() ## showing the first 5 rows of the duplicates ro
```

Out[20]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	tenure	work_
396	0.46	0.57	2	139	3	
866	0.41	0.46	2	128	3	
1317	0.37	0.51	2	127	3	
1368	0.41	0.52	2	132	3	
1461	0.42	0.53	2	142	3	

Check outliers:

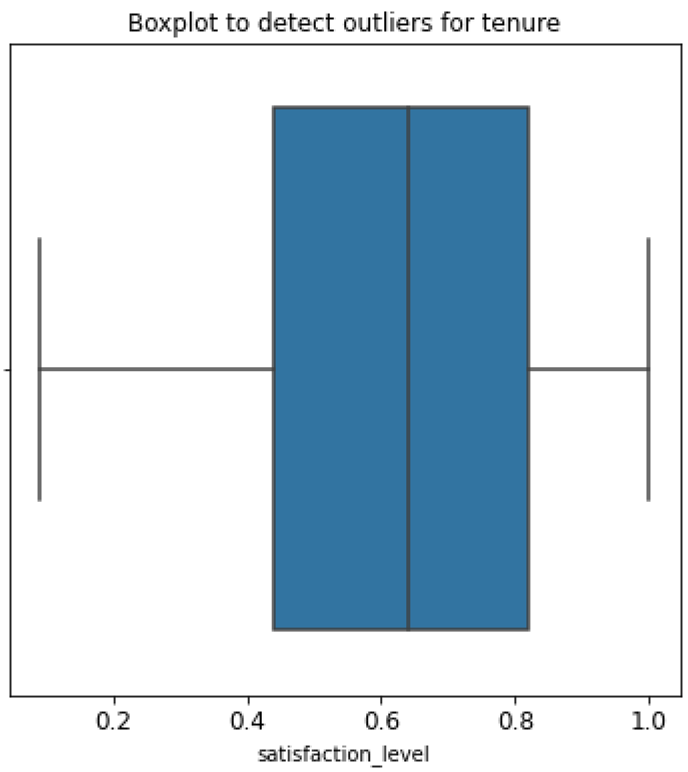
In [29]:
```
## Let's check outliers in our dataset:
## lets bring the descriptive analysis of our data to start
data.describe()
```

Out[29]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenu
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.0000(
mean	0.612834	0.716102	3.803054	201.050337	3.4982:
std	0.248631	0.171169	1.232592	49.943099	1.4601:
min	0.090000	0.360000	2.000000	96.000000	2.0000(
25%	0.440000	0.560000	3.000000	156.000000	3.0000(
50%	0.640000	0.720000	4.000000	200.000000	3.0000(
75%	0.820000	0.870000	5.000000	245.000000	4.0000(
max	1.000000	1.000000	7.000000	310.000000	10.0000(

In [32]:
```
## Let's create a boxplot to visulaize the outliers
## Satisfaction Level
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers in satisfaction level', fontsize
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['satisfaction_level'])
plt.show
```

Out[32]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

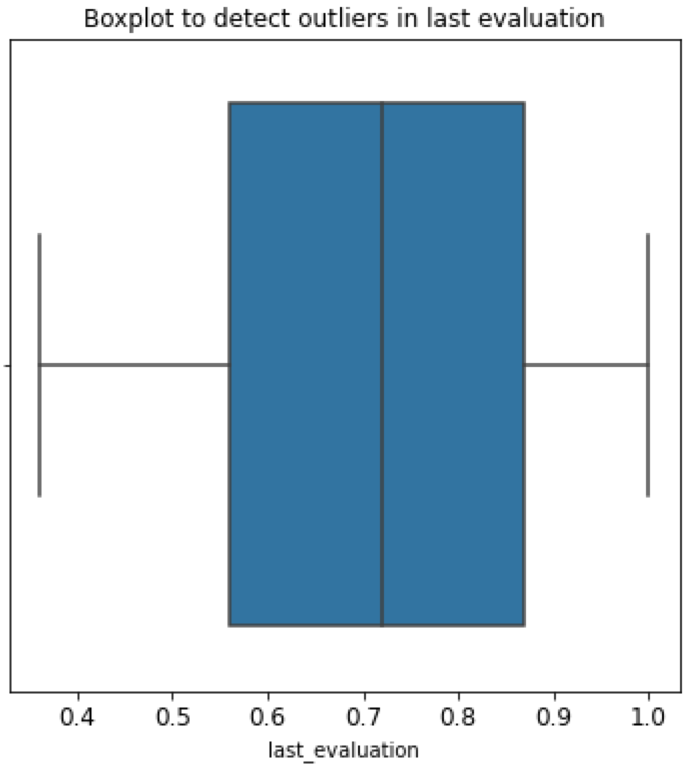


In [34]:

```
## lets visulaize the last evaluation

plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers in last evaluation', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['last_evaluation'])
plt.show
```

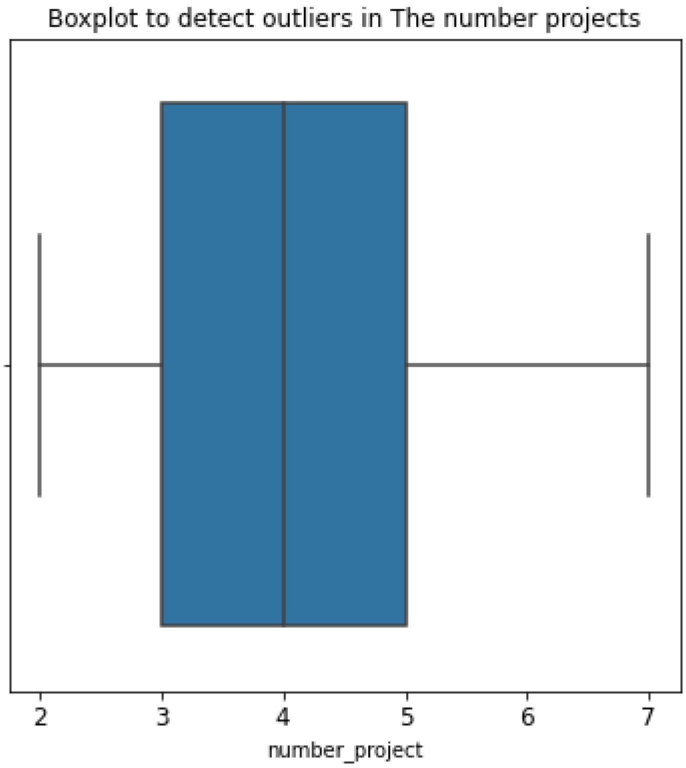
Out[34]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [35]:
```
## lets visulaize number_project
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers in The number projects',fontsize=
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['number_project'])
plt.show
```

Out[35]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

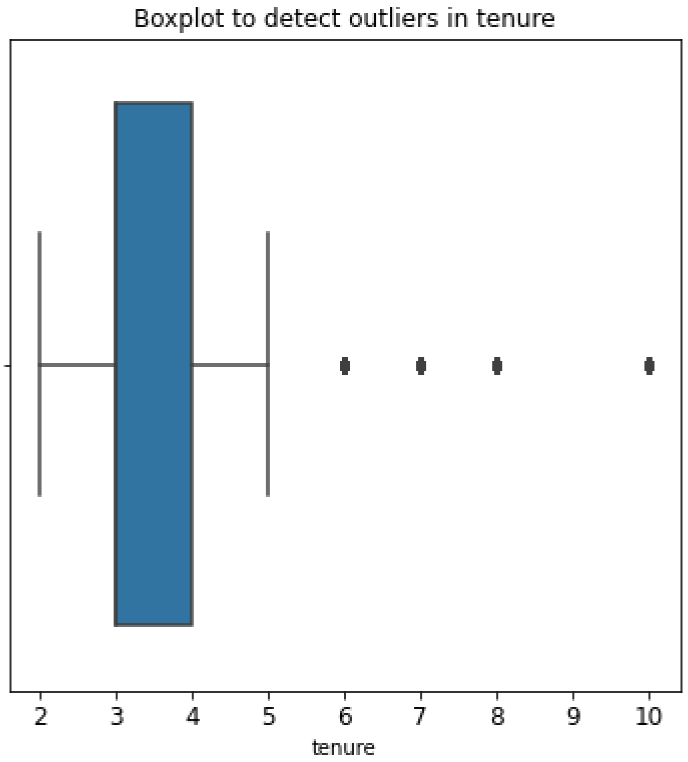


In [ ]:
```
## lets visulaize average monthly hours
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers in The averge monthly hours',font
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['average_monthly_hours'])
plt.show
```

In [37]:

```
## lets visulaize tenure
plt.figure(figsize=(6,6))
plt.title('Boxplot to detect outliers in tenure',fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
sns.boxplot(x=data['tenure'])
plt.show
```

Out[37]: <function matplotlib.pyplot.show(close=None, block=None)>



The boxplot above shows that there are outliers oin the `tenure` variable.

It would be helpful to investigate how many rows in the data contain outliers in the `tenure` column.

In [40]:
```
##Let's determine the number of rows containing outliers.
## Let's start by computing IQR(interquartile range)

percentile25 = data['tenure'].quantile(0.25)
percentile75 = data['tenure'].quantile(0.75)

iqr = percentile75 - percentile25

## lets define the upper limit and the lower limit for non-outliers val
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5* iqr
print("lower Limit:", lower_limit)
print("Upper Limit:", upper_limit)

## Let's identify the subset of the data containinf outliers in `tenure
outliers = data[(data['tenure'] > upper_limit) | (data['tenure'] < lowe

## let's identify how many rows containing outliers
print("Number of rows in the data containing outliers in tenure:",len(o

```

```
lower Limit: 1.5
Upper Limit: 5.5
Number of rows in the data containing outliers in tenure: 1282
```

Let's keep the outliers for now until determining which model to use and understand the model sensitivity to these outliers

Continuing EDA :

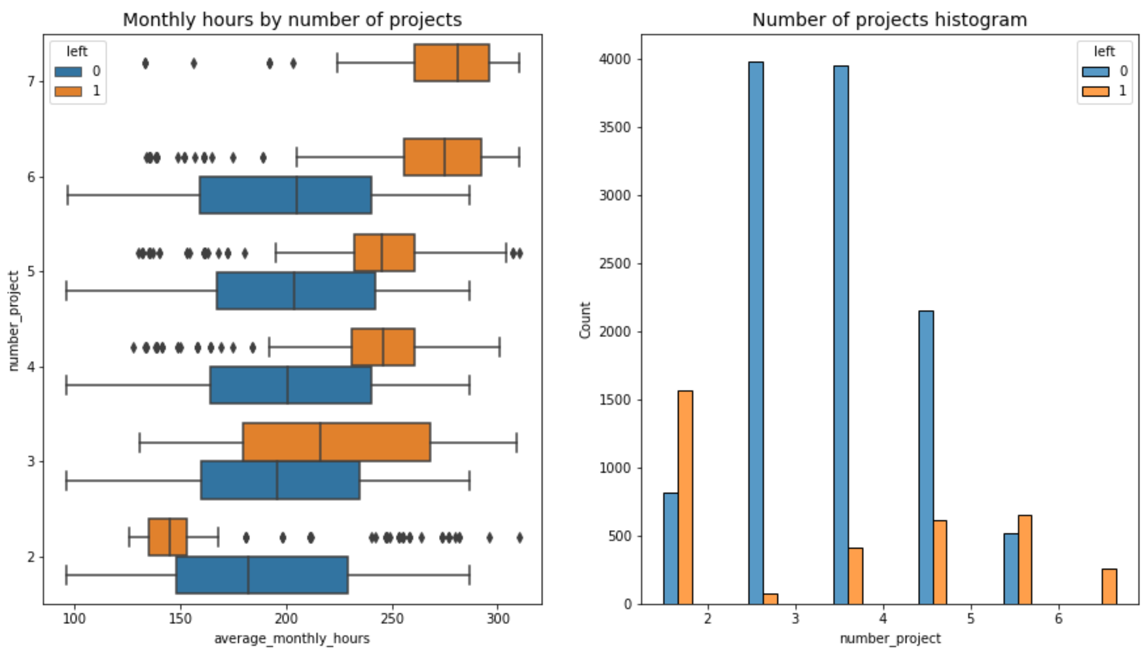
In [43]:
```
## Let's undersrtand how many employees left and what their representat
print(data['left'].value_counts())
print(data['left'].value_counts(normalize=True)*100)
```

```
0    11428
1     3571
Name: left, dtype: int64
0    76.191746
1    23.808254
Name: left, dtype: float64
```

Data visualizations:

In [61]:
```
## Let's create a boxplot showing 'average_monthly_hours' distributions
fig, ax = plt.subplots(1,2,figsize=(15,8))
sns.boxplot(data=data,x='average_monthly_hours',y='number_project',hue=
ax[0].invert_yaxis()
ax[0].set_title('Monthly hours by number of projects', fontsize=14)
## Now let's creat a histogram to visualize the distribution of number_
sns.histplot(data=data,x='number_project',hue='left', multiple='dodge',
ax[1].set_title("Number of projects histogram", fontsize =14)

plt.show

```

Out[61]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Notes:

It's natural that people work on more projects tend to work longer hours, this appears to be the case here, however a few things stand out from this plot.

1. There are two groups pf employees who left the company: (A) those who worked considerably less than their peers with the same number of projects, and (B) who worked much more. group (A) might be the people that who are serving their contract notice period and they were assigned to fewer hours.
2. The optimal number of projects for employees to work on seems to be 3-4, the ratio of left/stayed is very small.
3. If the employee should work 40 hours/week and 166.67 hours/month, the mean average of monthly hours is 201 and some of the employees worked 301 hours, it seems that the employees are overworked.
4. Every employee with project more than 6 left the company.

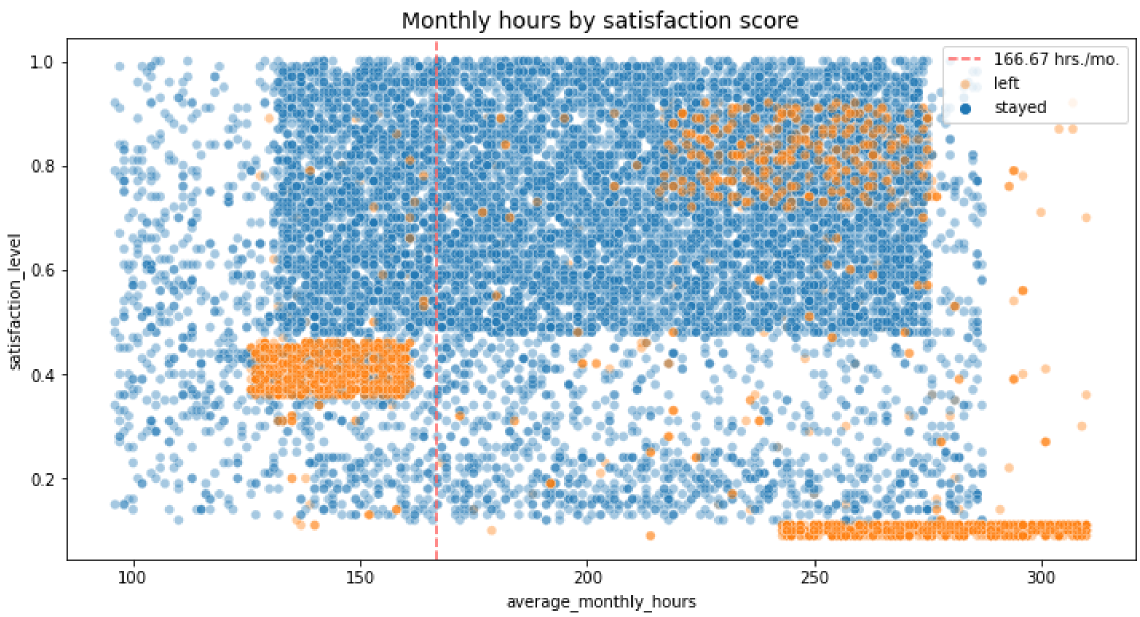
In [62]:
```
## Let confirm if all the employees with 7 prjects left the company.

data[data['number_project'] == 7]['left'].value_counts()
```

Out[62]:
```
1    256
Name: left, dtype: int64
```

This confirms that all employees with 7 projects left the company

In [65]:
```
##Let's examine the average monthly hours Vs the employee satisfaction

#Let's create scatterplot of the avergae monthly hours vs the employee
plt.figure(figsize=(12,6))
sns.scatterplot(data=data,x='average_monthly_hours',y='satisfaction_lev
plt.axvline(x=166.67,color = '#ff6361',label='166.67 hrs./mo.',ls='--')
plt.legend(labels=['166.67 hrs./mo.','left','stayed'])
plt.title('Monthly hours by satisfaction score',fontsize=14)
plt.show
```

Out[65]: `<function matplotlib.pyplot.show(close=None, block=None)>`

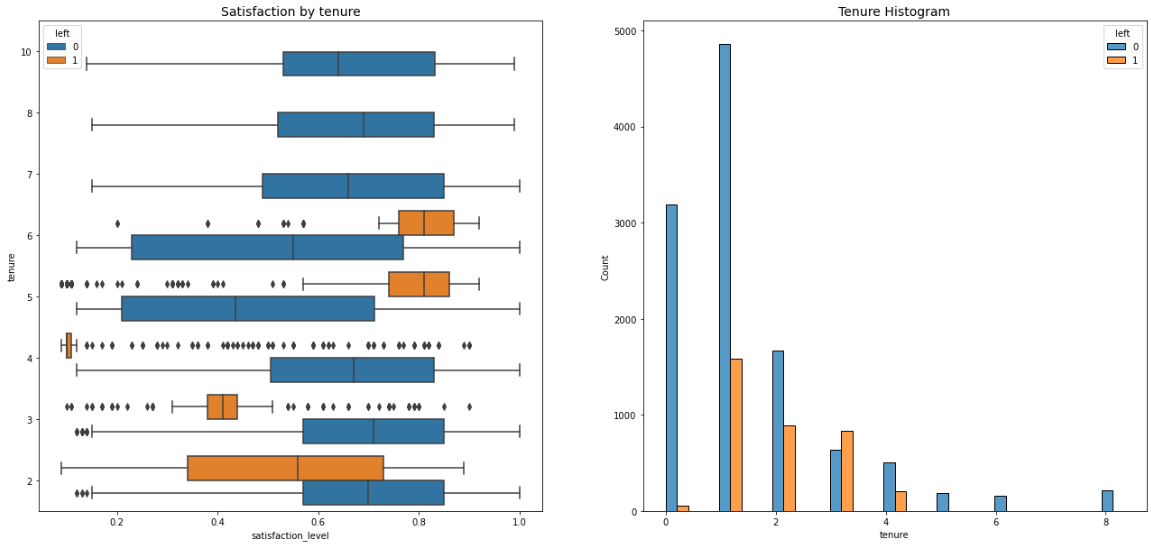


Notes:

1. From the above plot notice that a group of employees worked between 230 -330 hours/month and this is more that the average working hours, this could be the reason for their satisfaction level.
2. The plot also shows that there is a gorup of employees who worked minimum hours comparing to their peers and yet they left, their satisfaction score is around 0.4.
3. Finally, a third group who have worked between 210 - 280 hours/monnth they have left but their satisfaction level is above 0.75.

In [71]:

```
## Let's visualize the satisfaction level vs the tenure.

#lets set the figure axis
fig, ax = plt.subplots(1,2, figsize=(22,10))

## lets create a boxplot showing the distributions of the satisfaction

sns.boxplot(data=data,x="satisfaction_level",y='tenure',hue='left',orie
ax[0].invert_yaxis()
ax[0].set_title('Satisfaction by tenure',fontsize=14)


##Lets create a histogram showing the distribution of tenure, comparing
sns.histplot(data=data,x='tenure',hue='left',multiple='dodge',shrink=5,
ax[1].set_title('Tenure Histogram',fontsize ='14')

plt.show()

```



Observations:

1. Employees with longer-tenure tends to stay and the have the same satisfaction level as those who newly joined the company.
2. Employees at 4 years tenure have unusual satisfaction score, it worth checking the company policies or any changes happened at 4 year mark.
3. The majority of employees who left worked few years and they have low satisfaction level.

In [72]:

```
## Let's calculate the mean a median satisfaction scores of employees wh
data.groupby(['left'])['satisfaction_level'].agg([np.mean,np.median])

```

Out[72]:

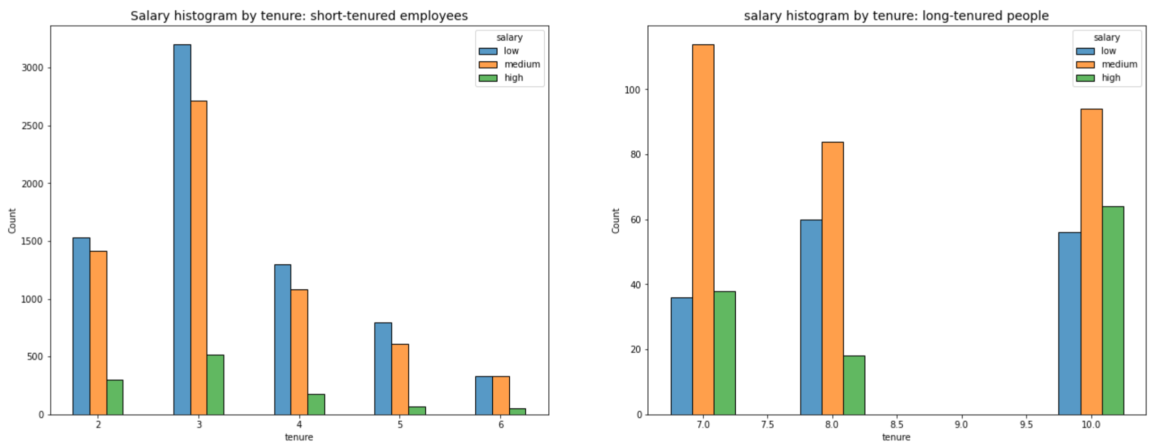
left	mean	median
0	0.666810	0.69
1	0.440098	0.41

Observations: The mean and median for those who left are lower than the score of the employees who stayed, Among the employees who stayed the mean is lower than the median which indicates that the satisfaction scores among those who stayed are skewed to the left.

In [78]:

```
## Let's examine the salary levels for different tenures.
# let's set the figure and axes
fig, ax = plt.subplots(1,2,figsize=(22,8))
# lets define short-tenured employees.
tenure_short = data[data['tenure'] < 7]
tenure_long = data[data['tenure']> 6]

#let's plot short_tenured histogram:
sns.histplot(data=tenure_short,x = 'tenure',hue='salary', hue_order=['l
ax[0].set_title('Salary histogram by tenure: short-tenured employees',f

#And the long_tenured employees:
sns.histplot(data=tenure_long,x='tenure', shrink=.5,hue='salary', hue_o
             discrete=1,ax=ax[1])
ax[1].set_title('salary histogram by tenure: long-tenured people',fonts
plt.show
```

Out[78]: `<function matplotlib.pyplot.show(close=None, block=None)>`



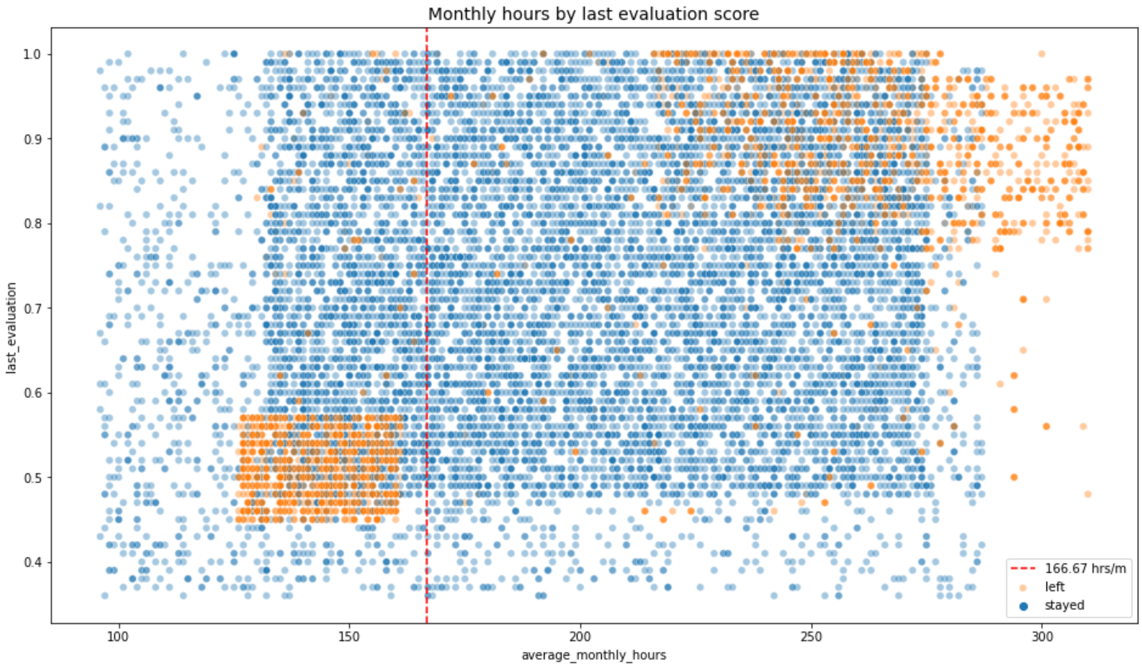
Observation: Being a long-tenured employee does not necessarily correlate with having a higher salary.

In [79]:

```python
## Let's look at the average monthly hours vs the evaluation scores:

plt.figure(figsize=(16,9))
sns.scatterplot(data=data,x='average_monthly_hours',y='last_evaluation'
plt.axvline(x=166.67,color='red',label='166.67 hr/m',ls='--')
plt.legend(labels=['166.67 hrs/m','left','stayed'])
plt.title('Monthly hours by last evaluation score',fontsize=14)
plt.show
```

Out[79]:

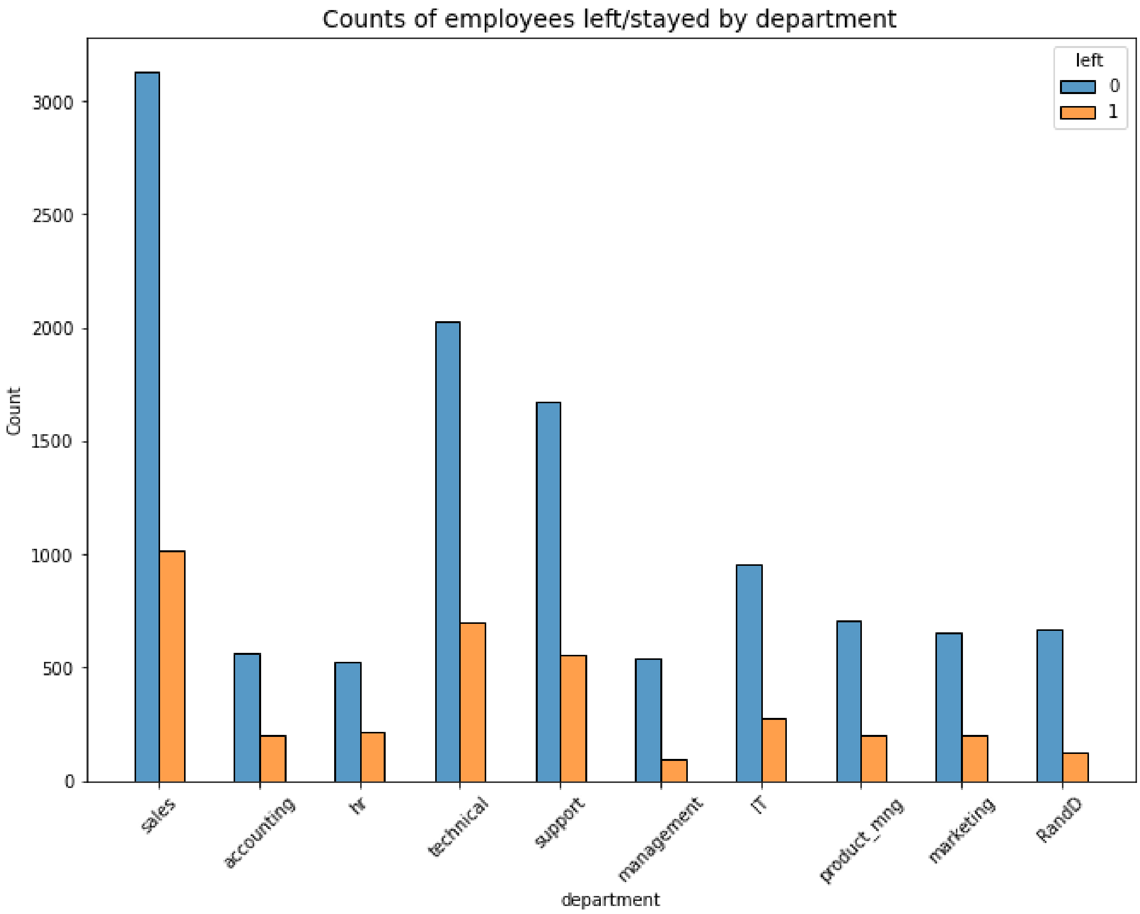
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Observations: 1. There seems to be a correlation between hours worked and evaluation score. 2. Most of the employees in this company work well over 167 hours per month.

In [80]:
```
## Now lets visualize the employees who left versus their department:
plt.figure(figsize=(11,8))
sns.histplot(data=data,x='department',hue='left',discrete=1,hue_order=[

plt.xticks(rotation=45)
plt.title('Counts of employees left/stayed by department',fontsize=14)
```

Out[80]: Text(0.5, 1.0, 'Counts of employees left/stayed by department')



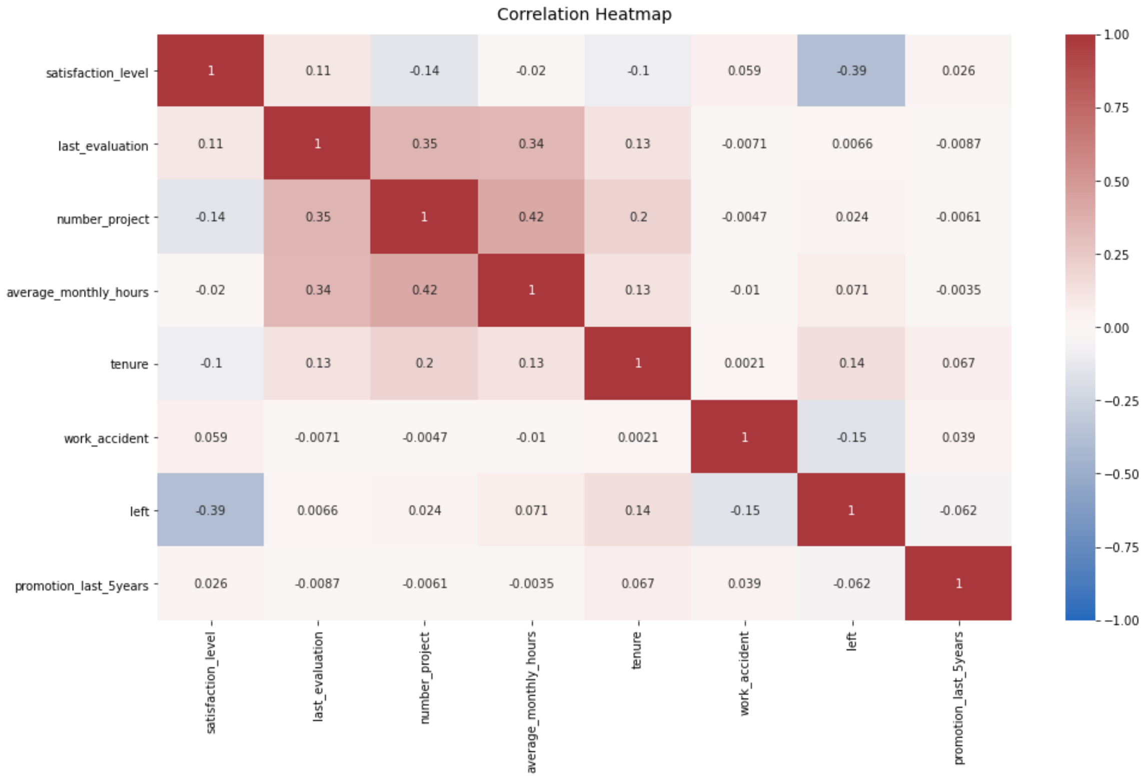
Observation: There doesn't seem to be any department that differs significantly in its proportion of employees who left to those who stayed

In [83]:

```
## Let's create a heatmap to check for a strong correlation between var
plt.figure(figsize=(16,9))
heatmap= sns.heatmap(data.corr(),vmin=-1,vmax=1,annot=True,cmap=sns.col
heatmap.set_title("Correlation Heatmap",fontdict={'fontsize':14},pad=12
plt.show
```

Out[83]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Observation: The correlation heatmap confirms that the number of projects, monthly hours, and evaluation scores all have some positive correlation with each other, and whether an employee leaves is negatively correlated with their satisfaction level.

Insights:

1. Leaving is tied to longer working hours, many projects, and generally lower satisfaction levels.
2. There's a sizeable group of employees at this company who are probably burned out.
3. It can be ungratifying to work long hours and not receive promotions or good evaluation scores.
4. It also appears that if the employee cross the 6 years tenure mark they tend to stay.

Notes: By examining the EDA insights and outcomes, we can start by chosing and developing the model.

Model Development:

Since the outcome variable is categorical, Lets develop a logisitc regression model and dicison tree model as well and compare how they performed.

Before splitting the data, lets encode the nonnumerical variables in the dataset, `department` and `salary`

Approach (A): Logistic Regression

In [88]:
```
##Lets copy the dataframe.
data_new = data.copy()

##Notice that salary is categorical but its not ordinal, there is a hie

data_new['salary'] = (data_new['salary'].astype('category').cat.set_cat

## And Lets dummy the department for modeling.
data_new= pd.get_dummies(data_new,drop_first=False)
data_new.head()
```

Out[88]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_ac
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	

In [89]:
```
## Since logistic regression is sensitive to outliers, lets remove the 
data_new = data_new[(data_new['tenure'] >= lower_limit) & (data_new['te

data_new.head()
```

Out[89]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_ac
0	0.38	0.53	2	157	3	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	
5	0.41	0.50	2	153	3	

In [90]:
```
## Now Lets Isolate the outcome variable and assign it to y.

y=data_new['left']
y.head()
```

Out[90]:
```
0    1
2    1
3    1
4    1
5    1
Name: left, dtype: int64
```

In [91]:
```
##Now lets select the features and assign it to X.

X = data_new.drop('left',axis=1)
X.head()
```

Out[91]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_a
0	0.38	0.53	2	157	3	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	
5	0.41	0.50	2	153	3	

In [98]:
```
## Here we would split hte data into training set and testing test, we 
## testing size would be 25%.
## lets import the required packages.
from sklearn.model_selection import train_test_split


X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,str

```

In [99]:
```
## Let's Construct the logistice regression.
## Lets Import the required packages.
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(random_state=42,max_iter =500)

## and fit the regression model

Log_f = log.fit(X_train,y_train)
```

In [101]:
```
## Lets test the model by using the model to get predictions.
y_pred = Log_f.predict(X_test)
```

In [ ]:
```
## Lets create a confusion matrix to visualize the results.
## Let's import the required packages (PS. lets import all the packages

from sklearn.metrics import accuracy_score, precision_score, recall_sco
f1_score, confusion_matrix, ConfusionMatrixDisplay, classification_repo
from sklearn.metrics import roc_auc_score, roc_curve

cm = confusion_matrix(y_test,y_pred,labels=Log_f.classes_)
cm_disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=Log
cm_disp.plot(values_format='')

plt.show()
```

Notice: The number of false negatives is higher than the number of false positives. This implies that the model is more conservative in predicting the positive class; it's more likely to miss positives (predict them as negatives, employees stayed), Lets check the outcome variable balanced in dataset

In [104]:
```
data_new['left'].value_counts(normalize=True)*100
```

Out[104]:
```
0    75.490268
1    24.509732
Name: left, dtype: float64
```

There is an approx. 75% to 25% split, which its not severly imbalance. lets continue evaluating the model.

In [105]:
```
## lets create a classification report
target_names = ['Predicated would not leave','Predicated would leave']
print(classification_report(y_test,y_pred,target_names=target_names))
```

```
                            precision    recall  f1-score   support

Predicated would not leave       0.86      0.91      0.88      2589
    Predicated would leave       0.66      0.56      0.60       841

                  accuracy                           0.82      3430
                 macro avg       0.76      0.73      0.74      3430
              weighted avg       0.81      0.82      0.81      3430
```

Observations:

1. The model is quite good at predicting the employees who would not leave (Class 0), as indicated by high precision, recall, and F1-score for this class.
2. The model struggles relatively more with predicting the employees who would leave (Class 1), which is evident from the lower recall and F1-score.
3. Improving the model could involve addressing the imbalance, perhaps by resampling the dataset

Approach (B): Tree-based Model

In [107]:
```
## Lets Isolate, build, fit and evluate a decision tree model.
## Prepare the dataset
##Lets copy the dataframe.
data_two = data.copy()

##Notice that salary is categorical but its not ordinal, there is a hie

data_two['salary'] = (data_two['salary'].astype('category').cat.set_cat

## And lets dummy the department for modeling.
data_two= pd.get_dummies(data_two,drop_first=False)
data_two.head()
```

Out[107]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_a
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	

In [108]:
```
## isolate, build and fit.
y = data_two['left']
y.shape
```

Out[108]: (14999,)

In [111]:
```
#Features.
X= data_two.drop('left',axis=1)
X.head()
```

Out[111]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	tenure	work_a
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	

In [112]:
```
## here we would split the data into training, validating and testing s
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,ran

```

In [113]:
```
##Lets import the necessary packages to build a tree based models.
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.tree import plot_tree
```

In [117]:
```
## Instantiate model & setting up cross-validated grid-search to search
tree= DecisionTreeClassifier(random_state=0)
##Assigning a dictionary of hyperparameters to search over

cv_params = {'max_depth':[4,6,8,None],'min_samples_leaf':[2,5,1],'min_s

##Assigning the scoring metrics

scoring = {'accuracy','precision','recall','f1','roc_auc'}

## Insintiate the GridSearch
tree1 = GridSearchCV(tree,cv_params,scoring=scoring,cv=5,refit='roc_auc

```

Fit the Tree on the training data.

In [118]:
```
%%time
tree1.fit(X_train,y_train)
```

```
Wall time: 4.26 s
```

Out[118]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             refit='roc_auc',
             scoring={'accuracy', 'roc_auc', 'precision', 'f1', 'recall'})
```

In [119]:
```
## Now lets identify the optimal values for the decision tree parameter
tree1.best_params_
```

Out[119]:
```
{'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

In [120]:
```
##Let's identify the best AUC score achieved by the decision tree model
tree1.best_score_
```

Out[120]:
```
0.9817669232989432
```

This score shows that the model can predict the employees who will leave very well.

Now lets write a function that will extract all the scores from the grid search.

In [121]:
```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): The model to be called in the output table
        model_object: a fit GridSearchCV object
        metric (string): precision, recall, f1, accuracy, or auc

    Returns a pandas df with the F1, recall, precision, accuracy, and a
    for the model with the best mean 'metric' score across all validati
    '''

    # Let's create dictionary that maps input metric to actual metric n
    metric_dict = {'auc': 'mean_test_roc_auc',
                   'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy'
                  }

    #Let's get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[met

    # Extract Accuracy, precision, recall, and f1 score from that row
    auc = best_estimator_results.mean_test_roc_auc
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

    #Let's create table of results
    table = pd.DataFrame()
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          'auc': [auc]
                         })

    return table
```

In [123]:
```
tree1_cv_results = make_results('Decision tree cv',tree1,'auc')
tree1_cv_results
```

Out[123]:

	model	precision	recall	F1	accuracy	auc
0	Decision tree cv	0.95304	0.934866	0.943722	0.973242	0.981767

All of these scores from the decision tree model are strong indicators of good model performance.

Note That decision trees can be vulnerable to overfitting. Random forest avoid overfitting by incorporating mutliple trees to make predictions, lets develop a Random forest model

In [131]:

```
rf = RandomForestClassifier(random_state=0)
## Let's assign a dictionary of hyperparameters to search over

cv_params = {'max_depth': [2,5,None],'max_features':[1.0],'max_samples'
             'min_samples_split':[2,3,4],'n_estimators':[300,500]}

##Let's instantiate GridSearch
rf1 = GridSearchCV(rf,cv_params,scoring=scoring,cv=5,refit='roc_auc')
```

In [132]:

```
%%time
##Let's fit the model.

rf1.fit(X_train,y_train)
```

```
C:\Users\engmo\anaconda3\lib\site-packages\sklearn\metrics\_classificati
on.py:1245: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 due to no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\engmo\anaconda3\lib\site-packages\sklearn\metrics\_classificati
on.py:1245: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 due to no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\engmo\anaconda3\lib\site-packages\sklearn\metrics\_classificati
on.py:1245: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 due to no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\engmo\anaconda3\lib\site-packages\sklearn\metrics\_classificati
on.py:1245: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 due to no predicted samples. Use `zero_division` parameter to
control this behavior.
```

In [136]:
```python
## Lets write a pickle functions to save and load the model results whe
import pickle
path = r'C:\Users\engmo\OneDrive\Desktop\Model Pickles'
def write_pickle(path, model_object, save_as:str):
    '''
    In:
        path:         path of folder where to save the pickle
        model_object: a model to pickle
        save_as:      filename for how to save the model

    Out: A call to pickle the model in the folder indicated
    '''

    with open(path + save_as + '.pickle', 'wb') as to_write:
        pickle.dump(model_object, to_write)

def read_pickle(path, saved_model_name:str):

    with open(path + saved_model_name + '.pickle', 'rb') as to_read:
        model = pickle.load(to_read)

    return model

```

In [138]:
```python
##lets save the model in local drive.
write_pickle(path, rf1, 'hr_rf1')
```

In [139]:
```python
## lets read the pickle into the environement
rf1 = read_pickle(path, 'hr_rf1')
```

In [140]:
```python
#Lets determine the best score
rf1.best_score_
```

Out[140]:
```
0.9906091306922387
```

In [141]:
```python
## Let's identify the best params
rf1.best_params_
```

Out[141]:
```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 3,
 'min_samples_split': 2,
 'n_estimators': 300}
```

In [142]:
```python
##lets gather all the scores:
rf1_cv_results = make_results('Random Forest CV',rf1,'auc')
print(tree1_cv_results)
print(rf1_cv_results)
```

```
              model  precision    recall        F1  accuracy       auc
0  Decision tree cv    0.95304  0.934866  0.943722  0.973242  0.981767
              model  precision    recall        F1  accuracy       auc
0  Random Forest CV   0.988597  0.925613  0.955976  0.979554  0.990609
```

The evaluation scores of the random forest model are better than those of the decision tree model, with the exception of recall (the recall score of the random forest model is lower, which is a negligible amount). This indicates that the random forest model mostly outperforms the decision tree model.

Finally, lets evaluate the model on the test data.

In [143]:
```
##Let's define a functions to get all the scores from the model's predi
def get_scores(model_name:str, model, X_test_data, y_test_data):
    '''
    Generate a table of test scores.

    In:
        model_name (string):  How you want your model to be named in th
        model:                A fit GridSearchCV object
        X_test_data:          numpy array of X_test data
        y_test_data:          numpy array of y_test data

    Out: pandas df of precision, recall, f1, accuracy, and AUC scores f
    '''

    preds = model.best_estimator_.predict(X_test_data)

    auc = roc_auc_score(y_test_data, preds)
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'f1': [f1],
                          'accuracy': [accuracy],
                          'AUC': [auc]
                         })

    return table
```

In [145]:
```
#predictions on test data
rf1_test_scores = get_scores('random forest test', rf1, X_test, y_test)
rf1_test_scores
```

Out[145]:

	model	precision	recall	f1	accuracy	AUC
0	random forest test	0.9807	0.935558	0.957597	0.9808	0.965002

This appears to be a strong model. a good indictive that it will perform good on the unseen data. We could stop here but there might be a data leakage, as explained earlier, For example the column average_monthly_hours could be a source of a leakage that the employee decided on leaving and therefore worked minimum hours.

Lets Alter the features in this model and compare the results.

In [146]:
```
#Lets create a feature called overworked by assigning the employees who
data_f = data_two.drop('satisfaction_level',axis=1)
```

In [147]:
```
data_f.head()
```

Out[147]:

	last_evaluation	number_project	average_monthly_hours	tenure	work_accident	left	prom
0	0.53	2	157	3	0	1	
1	0.86	5	262	6	0	1	
2	0.88	7	272	4	0	1	
3	0.87	5	223	5	0	1	
4	0.52	2	159	3	0	1	

In [148]:
```
#Lets add the new column.
data_f['overworked'] =data_f['average_monthly_hours']
## as stated earlier the normal working hours per month is 166.67 hrs/m
data_f['overworked'] = (data_f['overworked'] >175).astype(int)
data_f['overworked'].head()
```

Out[148]:
```
0    0
1    1
2    1
3    1
4    0
Name: overworked, dtype: int32
```

In [149]:
```
## Let's drop the average_monthly_hours column.
data_f.drop('average_monthly_hours', axis=1,inplace=True)
data_f.head()
```

Out[149]:

	last_evaluation	number_project	tenure	work_accident	left	promotion_last_5years	salary
0	0.53	2	3	0	1	0	0
1	0.86	5	6	0	1	0	1
2	0.88	7	4	0	1	0	1
3	0.87	5	5	0	1	0	0
4	0.52	2	3	0	1	0	0

In [150]:
```
#Let's Isolate the outcome variable
y = data_f['left']

#selecting the features
X= data_f.drop('left',axis=1)

```

In [151]:
```
#Lets split data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,str
```

In [152]:
```
## Tree based, instantiate the model
tree = DecisionTreeClassifier(random_state=0)

#Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth':[4, 6, 8, None],
             'min_samples_leaf': [2, 5, 1],
             'min_samples_split': [2, 4, 6]
             }

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
tree2 = GridSearchCV(tree, cv_params, scoring=scoring, cv=4, refit='roc
```

In [153]:
```
%%time
tree2.fit(X_train, y_train)
```

```
Wall time: 2.96 s
```

Out[153]:
```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [4, 6, 8, None],
                         'min_samples_leaf': [2, 5, 1],
                         'min_samples_split': [2, 4, 6]},
             refit='roc_auc',
             scoring={'accuracy', 'roc_auc', 'precision', 'f1', 'recall'})
```

In [154]:
```
#Let's check best params
tree2.best_params_
```

Out[154]:
```
{'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

In [155]:
```
#Let's check best AUC score on CV
tree2.best_score_
```

Out[155]:
```
0.9707127603293837
```

This model performs very well, even without satisfaction levels and detailed hours worked data.

Next, let's check the other scores.

In [156]:
```
# Get all CV scores
tree2_cv_results = make_results('decision tree2 cv', tree2, 'auc')
print(tree1_cv_results)
print(tree2_cv_results)
```

```
              model  precision    recall        F1  accuracy       auc
0  Decision tree cv    0.95304  0.934866  0.943722  0.973242  0.981767
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv    0.91728  0.891332  0.904003   0.95493  0.970713
```

Scores fell. That's to be expected given fewer features were taken into account in this round of the model. Still, the scores are very good.

In [157]:
```
## Let's see the instantiate the random forest.
# Instantiate model
rf = RandomForestClassifier(random_state=0)

# Assign a dictionary of hyperparameters to search over
cv_params = {'max_depth': [3,5, None],
             'max_features': [1.0],
             'max_samples': [0.7, 1.0],
             'min_samples_leaf': [1,2,3],
             'min_samples_split': [2,3,4],
             'n_estimators': [300, 500],
             }

# Assign a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1', 'roc_auc'}

# Instantiate GridSearch
rf2 = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='roc_auc
```

In [158]:
```
%%time
rf2.fit(X_train, y_train)
```

```
C:\Users\engmo\anaconda3\lib\site-packages\sklearn\model_selection\_vali
dation.py:610: FitFailedWarning: Estimator fit failed. The score on this
train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\engmo\anaconda3\lib\site-packages\sklearn\model_selecti
on\_validation.py", line 593, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\engmo\anaconda3\lib\site-packages\sklearn\ensemble\_for
est.py", line 343, in fit
    n_samples_bootstrap = _get_n_samples_bootstrap(
  File "C:\Users\engmo\anaconda3\lib\site-packages\sklearn\ensemble\_for
est.py", line 110, in _get_n_samples_bootstrap
    raise ValueError(msg.format(max_samples))
ValueError: `max_samples` must be in range (0, 1) but got value 1.0

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\engmo\anaconda3\lib\site-packages\sklearn\model_selection\_vali
dation.py:610: FitFailedWarning: Estimator fit failed. The score on this
train-test partition for these parameters will be set to nan. Details:
```

In [159]:
```
# Write pickle
write_pickle(path, rf2, 'hr_rf2')
```

In [160]:
```
# Read in pickle
rf2 = read_pickle(path, 'hr_rf2')
```

In [161]:
```
# Check best params
rf2.best_params_
```

Out[161]:
```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 500}
```

In [162]:
```
# Check best AUC score on CV
rf2.best_score_
```

Out[162]: 0.9800155462940825

In [163]:
```
# Get all CV scores
rf2_cv_results = make_results('random forest2 cv', rf2, 'auc')
print(tree2_cv_results)
print(rf2_cv_results)
```

```
               model  precision    recall        F1  accuracy       auc
0  decision tree2 cv    0.91728  0.891332  0.904003   0.95493  0.970713
               model  precision    recall        F1  accuracy       auc
0  random forest2 cv   0.933895  0.907015  0.920104  0.962486  0.980016
```

Again, the scores dropped slightly, but the random forest performs better than the decision tree if using AUC as the deciding metric.

Score the champion model on the test set now.

In [164]:
```
# Get predictions on test data
rf2_test_scores = get_scores('random forest2 test', rf2, X_test, y_test
rf2_test_scores
```

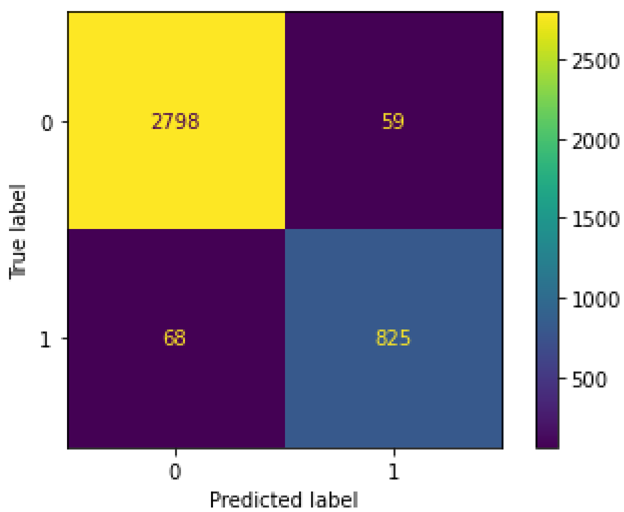
Out[164]:

	model	precision	recall	f1	accuracy	AUC
0	random forest2 test	0.933258	0.923852	0.928531	0.966133	0.951601

This seems to be a stable, well-performing final model.

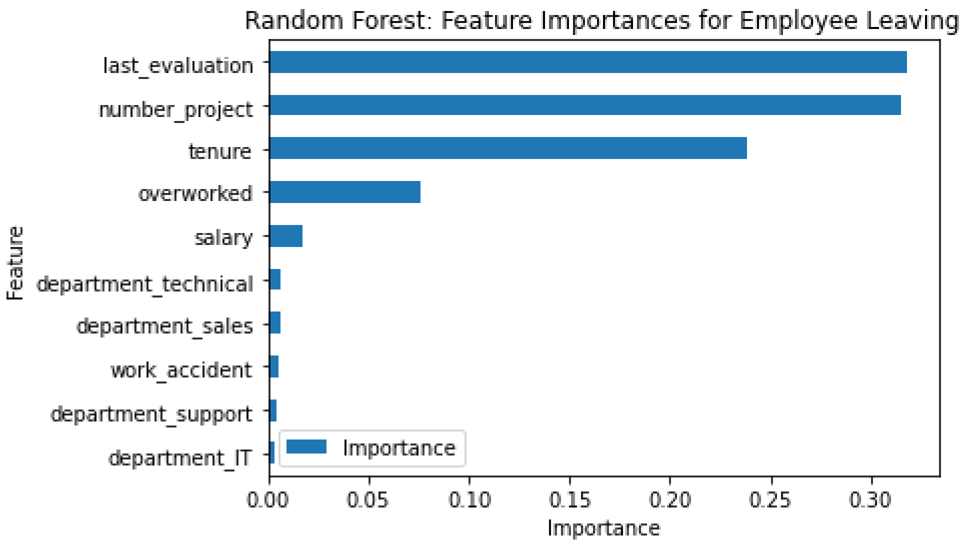
Let's plot a confusion matrix to visualize how well it predicts on the test set.

In [165]:
```
# Generate array of values for confusion matrix
preds = rf2.best_estimator_.predict(X_test)
cm = confusion_matrix(y_test, preds, labels=rf2.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=rf2.classes_)
disp.plot(values_format='');
```



For exploratory purpose,Let's inspect the most important features in the random forest model.

In [166]:
```
# Get feature importances
feat_impt = rf2.best_estimator_.feature_importances_

# Get indices of top 10 features
ind = np.argpartition(rf2.best_estimator_.feature_importances_, -10)[-1

# Get column labels of top 10 features
feat = X.columns[ind]

# Filter `feat_impt` to consist of top 10 feature importances
feat_impt = feat_impt[ind]

y_df = pd.DataFrame({"Feature":feat,"Importance":feat_impt})
y_sort_df = y_df.sort_values("Importance")
fig = plt.figure()
ax1 = fig.add_subplot(111)

y_sort_df.plot(kind='barh',ax=ax1,x="Feature",y="Importance")

ax1.set_title("Random Forest: Feature Importances for Employee Leaving"
ax1.set_ylabel("Feature")
ax1.set_xlabel("Importance")

plt.show()
```



The plot above shows that in this random forest model, `last_evaluation` , `number_project` , `tenure` , and `overworked` have the highest importance, in that order. These variables are most helpful in predicting the outcome variable, `left`

Summary of model results:

- Logistic Regression:

The logistic regression model achieved precision of 81%, recall of 82%, f1-score of 81% (all weighted averages), and accuracy of 82%, on the test set.

- Tree-based Machine Learning: (Decision Tree & Random Forest)

After conducting feature engineering, the decision tree model achieved precision of 93.0%, recall of 92%, f1-score of 92%, and accuracy of 96%, on the test set. The random forest modestly outperformed the decision tree model.

Conclusion: The models and the feature importances extracted from the models confirm that employees at the company are overworked(higher hours per month & number of projects).

Recommendations:

*Cap the number of projects that employees can work on. *Consider promoting employees who have been with the company for atleast four years, or conduct further investigation about why four-year tenured employees are so dissatisfied. *Either reward employees for working longer hours, or don't require them to do so. *If employees aren't familiar with the company's overtime pay policies, inform them about this. If the expectations around workload and time off aren't explicit, make them clear. *Hold company-wide and within-team discussions to understand and address the company work culture, across the board and in specific contexts. *High evaluation scores should not be reserved for employees who work 200+ hours per month. Consider a proportionate scale for rewarding employees who contribute more/put in more effort.

Next Steps:

It may be justified to still have some concern about data leakage. It could be prudent to consider how predictions change when last_evaluation is removed from the data. It's possible that evaluations aren't performed very frequently, in which case it would be useful to be able to predict employee retention without this feature. It's also possible that the evaluation score determines whether an employee leaves or stays, in which case it could be useful to pivot and try to predict performance score. The same could be said for satisfaction score.

In [ ]:

```

```